Determination of File Type and Attributes Including Endian-ness of an Overset Mesh Related File

Shishir Pandya

NASA Ames Research Center, M/S 258-2, Moffett Field, CA 94035

I. Introduction

The overset mesh approach has a 30 year history in which it has introduced several file formats. While all formats deal with describing mesh, flow, or grid-to-grid communications, some have morphed over time making the task of input/output of overset data difficult. While file extensions are used to surmount the challenges keeping track of the file type (e.g. x.save for a grid file vs. q.save for a solution file), the file attributes such as ASCII or unformatted, single or multiple grid, iblanking, single or double precision, 2D or 3D can not be determined easily. Furthermore, the computer's byte order becomes an issue since both little endian and big endian byte orders are in use on different machine architectures. The use of a file generated on a big endian machine by a person using a little endian machine can pose a challenge that is only solved by compiler options which allow the use of the other byte order by converting each number as it is read. A library of functions that can meet the challenge of being able to deal with any file type with any attribute, and with any byte order is necessary.

The software libraries described here are expressly written to solve this problem in an efficient manner.

II. Approach

While a majority of the CGT code is written in Fortran90 or Fortran77, portions of it (e.g. the OVERGRID GUI) are also written in C. Thus, it was deemed necessary to have I/O libraries in each language. While the libraries in Fortran are called the FFIO functions (all functions, subroutines, and file names begin with FFIO), the C functions, and files are prepended with "over". All I/O library files are contained in the lib directory.

Without delving deeply into the details of how to implement these functions in each language, we describe the basic concepts in this document. Note that while Fortran has functions for reading Fortran unformatted files, the "unformatted file I/O" mode (FORM=UNFORMATTED) does not allow the user to access the record marker. For this reason, all of the FFIO functions and subroutines use "stream I/O" mode. Similarly, since the C language does not have a Fortran unformatted file reading capability, we read the record markers directly. in both cases, the record markers are interpreted to gain an understanding of what the file attributes are.

A. ASCII or Unformatted?

We know that only grid, Plot3d solution, tri, and triq files can be ASCII. We also know that the first line in these files has anywhere from 1 to 5 integers. We now open the file with the assumption that the file is ASCII (In fortran, this is done with FORM=FORMATTED in the open statement). We now attempt to read the first integer from the file. The read statement is given the option to catch an error in this read with the "ERR=line#" option. If the file is ASCII, this attempt will be successful. However, if it is unformatted, the code will throw an error and go to the specified line number where we can set a flag to indicate that the file contains unformatted data.

B. Big or little endian?

1. Machine endian

A machine can be big or little endian. Which type the machine is can be determined by examining the order in which the bytes are written. So we start by initializing an integer, $j = 1$ to load 1 in the right-most byte of the

integer. Looking at it in bits it is

$$0000\ 0000\ 0000\ 0001 \tag{1}$$

Then using the left shift operator $k = 2 << 8$, we create the next integer

$$0000\ 0000\ 0010\ 0000 \tag{2}$$

Combining this with the j in Eqn. 1 with an inclusive OR ($j = j \mid k$), we get

$$0000\ 0000\ 0010\ 0001 \tag{3}$$

Thus loading the number 2 in the third byte of the integer. Similarly, we load the number 3 in the second byte: $j = j \mid 3 << 16$, and 4 in the first byte: $j = j \mid 4 << 24$. So that we get

$$0100\ 0011\ 0010\ 0001 \tag{4}$$

Now, we can interrogate the "first" byte in the integer (using (char *)&j because in c, (char *) is 1 byte and &j is the address of j) to see if it has a 1 or a 4 in it. If it interprets the left-most byte as the first byte (reads 4), its a big-endian machine. If instead the interpretation is that the right most byte is the first byte (reads 1), then its a little-endian machine.

2. File Endian

Most overset mesh files store just the number of grids or X-rays in the first record. This means that the first record marker = 4. The other possibilities are 2 ints for a 2D single grid (or solution) file or a tri file, 3 ints for a 3D single grid (or solution) file or a triq file, 4 ints in an XINTOUT file, and ngrids ints in an INTOUT file. A possible variation is that the solution files is an Overflow solution file and has the *nq* and *nqc* after the grid dimensions. In this case, there are 4 ints for 2D and 5 ints for 3D. Thus, an INTOUT file has the largest first record. If the INTOUT file has less than 4,194,303 grids, than the record size (16,777,215) can fit into the last 3 bytes of the integer and the first byte is zero. So, making the assumption that the first byte of the record marker is zero for a file of interest is sufficient until our grid systems get so large that the number of grids exceeds the 4.2 million mark. With this assumption, if the first byte of the record marker is zero, the file is big-endian; if not, it is little-endian.

C. File Type and Attributes Determination

To determine type and attributes, we must look at what are the possible types and attributes and what we can check in order to determine them. The possible file types are: Plot3d grid, Plot3d Q, Plot3d function, Overflow Q, Overflow-2 Q, Overflow-2 Q with 2nd order restart info, X-ray, INTOUT, XINTOUT, Cart3d tri and Cart3d triq. The possible attributes are: ASCII or unformatted, single or multiple grid, iblanking, single or double precision, 2D or 3D.

Because the records in an unformatted file defer from file to file it is possible to examine the records and what is contained in them to determine the type and attribute of the file. However, the files also defer in number of records and the sizes of the records. Thus, it is sufficient to look at the number and size of records to determine all of the information about the file. Table 1 lists the number of records and record sizes for each grid type and attribute. In this table, *ng* is the number of grids, *nx* is the number of X-rays, *np* is the number of points in the mesh, and $npq = np \times nq$ where *nq* is the number of flow variables in the file.

Using this table, one can imagine many choices of how to examine the number of records and record sizes. However, knowing these two pieces of information is not enough since we also need to know the number of grids or X-rays and the number of points in the first grid. Knowing the number of q variables is also useful for determining attributes of a Q file. This basic information must be read from the first 2 records of the file. Thus, it is difficult to avoid not reading the first 2 records.

From table 2 we also know that upto 4 record sizes may be needed to determine all attributes. Thus, we first count the number of records. In order to count the number of records we use the fact that the record marker stores the number of bytes in a record. We use this information to skip reading the record. Reading the ending record marker gives us a check that we are in the correct position in the file. We repeat this, until we run out of records. For the first 4 records, we also store the record size for later use in determining the attributes of the file.

Armed with this information, we begin the process of determining the file type. We know that the first record always has integers. Thus, we can look at the size of the first record to separate file types. For example, if the size

of the first record is 4 bytes, which is 1 integer, than it can be seen from the tables that it may be one of Plot3d grid, Plot3d Q, Overflow Q, Plot3d function, or X-ray files. If we now read the first record which contains the number of grids and look at the number of records, we can determine that if we have 3 grids, 5 records means that it is a grid file, 8 or 9 records indicates a Q file, between 10 and 14 records indicated an x-ray file, and 14 or 15 records indicates Overflow-2 Q with Q^{n-1}.

Table 1. Number of records and record sizes for various plot3d grid types and attributes.

File			Number	Rec.1	Rec.2		Rec.3	
Type	**ID**	**Attributes**	of records	ints	ints	reals	ints	reals
Plot3d Whole	0	SG, 2D	2	2	0	$2n_p$		
Plot3d Whole	0	SG, 2D, ib	2	2	$n_p p$	$2n_p$		
Plot3d Whole	0	SG, 3D	2	3	0	$3n_p$		
Plot3d Whole	0	SG, 3D, ib	2	3	n_p	$3n_p$		
Plot3d Whole	0	MG, 2D	$2+ng$	1	$2n_g$	0	0	$2n_p$
Plot3d Whole	0	MG, 2D,ib	$2+ng$	1	$2ng$	0	n_p	$2n_p$
Plot3d Whole	0	MG, 3D	$2+ng$	1	$3ng$	0	0	3np
Plot3d Whole	0	MG, 3D,ib	$2+ng$	1	$3ng$	0	np	3np
Plot3d Planes	1	SG, 3D	$1+L_{max}$	3	0	$3j_{max}k_{max}$		
Plot3d Planes	1	SG, 3D, ib	$1+L_{max}$	3	$j_{max}k_{max}$	$3j_{max}k_{max}$		
Plot3d Planes	1	MG, 3D	$2+\sum_{g=1}^{ng} L_{max}$	1	$3ng$	0	0	3np
Plot3d Planes	1	MG, 3D,ib	$2+\sum_{g=1}^{ng} L_{max}$	1	$3ng$	0	np	3np

Let's assume that we have a grid file with 3 grids, then we can immediately say that since it has only 1 integer in the first record, it is a multiple grid file. Since the 2nd record of a multiple-grid grid file contains the jmax, kmax, lmax (grid dimensions) information, we can use the size of the 2nd record to determine if the file is 2D,or 3D. The 3rd record of a grid file contains both the XYZs of the grid and the iblanks. The size of this record can be used to determine if the iblanks are in the file as well as whether the file is single or double precision.

Similar logic must be used to determine all file types and attributes. The reason that some times we can determine everything from the first 2 records and sometimes we need as many as 4, has to do with the assumptions about the files and the fact that we must know the size of the first record that contains non-integers to figure out whether the file is single or double precision.

Tables 1, 2, and 3 contain a complete list of all supported file types. It contains all information necessary to determine file type including the number of records and the size of each record.

D. Attribute Determination for a Specific File Type

When the file type is known (e.g. the files is a Plot3d grid file), the record-count often provides very little further information. In these cases, the sizes of the first 2 to 4 records provide the necessary information to determine what the file attributes are. Thus, specific algorithms are provided to determine attributes of heavily used file types (e.g. grid, and solution files). These type-specific routines tend to be much more efficient and are preferred when file type is known.

The triangulation formats do not use the record counts or record sizes for file type determination. The types then are determined simply based on the file extension in the filename.

E. Conflicts

In a few cases, conflicts arise. In these cases, it is not possible to automatically determine file type. So, assumptions are made that generally favor plot3d whole format for grid files, and an overflow standard Q format for Q files.

One example is the single grid (SG) format when $L_{max} = 1$. Here, the number of records is 2. Since the first record (Rec. 1) is always 3 integers or 12 bytes, the second record is used to tell if the grid was written in whole

[a] 7 reals, 1 integer and then 5 more reals
[b] 6 reals, 1 integer and then 5 more reals
[c] 7 reals, 1 integer and then 8 more reals

Table 2. Number of records and record sizes for various Q file types and attributes.

File			Number	Rec.1	Rec.2		Rec.3		Rec.4
Type	**ID**	**Atrributes**	of records	ints	ints	reals	ints	reals	reals
Plot3d Function	9	SG, 2D	2	3	0	npq			
Plot3d Function	9	SG, 3D	2	4	0	npq			
Plot3d Function	9	MG, 3D	$2+ng$	1	$4ng$	0	0	npq	
Plot3d Q	10	SG, 2D	3	2			0	10np	
Plot3d Q	10	SG, 3D	3	3			0	15np	
Plot3d Q	10	MG, 2D	$2+2ng$	1	$2ng$	0			10np
Plot3d Q	10	MG, 3D	$2+2ng$	1	$3ng$	0			15np
Overflow Q	11	SG, 2D	3	4	1	$7+5^{a}$	0	npq	
Overflow Q	11	SG, 3D	3	5	1	$7+5^{a}$	0	npq	
Overflow Q	11	MG, 2D	$2+2ng$	1	$2+2ng$	0	1	$7+5^{a}$	npq
Overflow Q ($ver. < 1.7e$)	11	MG, 3D	$2+2ng$	1	$2+3ng$	0	1	$6+5^{b}$	npq
Overflow Q ($1.7e \le ver. < 2.0p$)	12	MG, 3D	$2+2ng$	1	$2+3ng$	0	1	$7+5^{a}$	npq
Overflow-2 Q	13	SG, 3D	3[+1]	5	1	$7+8^{c}$	0	npq	
Overflow-2 Q ($ver. \ge 2.0p$)	13	MG, 3D	$2+2ng$[+1]	1	$2+3ng$	0	1	$7+8^{c}$	npq
Overflow-2 Q+Q^{n-1}	14	SG, 3D	5[+1]	5	1	$7+8^{c}$	0	npq	
Overflow-2 Q+Q^{n-1}	14	MG, 3D	$2+4ng$[+1]	1	$2+3ng$	0	1	$7+8^{b}$	npq
Overflow-2 Q-Average	15	MG, 3D	$2+2ng$	1	$2+3ng$	0	1	$7+8^{c}$	npq

Table 3. Number of records and record sizes for various other file types and attributes.

File			Number	Rec.1	Rec.2		Rec.3	
Type	**ID**	**Attributes**	of records	ints	ints	reals	ints	reals
X-ray	20	MG, 3D	$2+2nx \le n \le 2+4nx$	1	$3nx$	0	0	6
INTOUT	21	MG, 3D	1+2ng	ng	$7ib$	$3ib$		
XINTOUT (Pegasus)	22	MG, 3D	7ng	7	$3ii$	$3ii$		
XINTOUT (Overflow)	22	MG, 3D	7ng	4	$3ii$	$3ii$		
.tri	30	3D, SP	4	2				
i.tri	31	3D, SP	4	2				
TRI-Extended	32	3D, SP	6	4				
TRIQ	33	3D, SP	5	3				
TRIX(Unsupportred)	34							
FST	35							
UCD	36							
STL	37							
Tecplot	38							
FRO	39							

or planes format. This record holds the x, y, z information. The number of items in the record is $3np$, where $np = J_{max}K_{max}L_{max}$. When iblanks are included, there is an additional np integers. Since for plane formats, $np = J_{max}K_{max}$, there is a conflict when $L_{max} = 1$.

A similar example for multiple grid format: The number of records is $2+ng$ for whole format, and $2+ng*L_{max}$ for planes format, where ng is the number of grids. When $L_{max} = 1$, these are equal. So we look to the record sizes. The first record is always the number of grids and is thus identical between whole and planes formats. The second record is also identical. The third and subsequent records are np reals which again is the same when $L_{max} = 1$.

III. Routine Layout

There are 4 main files that provide a majority of the I/O capability in CGT's lib directory. The FFIOEndian.F90 file provide all endian determination, handling, and conversion routines. FFIOFileType.F90 file contain the routines that determine file type and attributes. FFIOReadRec.F90 file contain the routines for reading all file types listed in tables 1, 2, and 3 for any attribute. The corresponding writing capability is provided in the FFIOWriteRec.F90 file. Finally, the FFIOInterpret.F90 file contains routines for reporting the file attributes to the user.

In previous versions, there was an equal effort to support the same capability in c. However, we have opted to remove the c code, and now provide a C API for the fortran routines that are used by c codes. Specifically, the only c code now using the FFIO routines is overgrid, and thus the calls to the API are made from gui/og.c.